UNIVERSITÀ DI PISA

SCUOLA SUPERIORE SANT'ANNA PISA

MASTER DEGREE in EMBEDDED COMPUTING SYSTEMS
A.Y. 2017 - 2018

Schedulability Analysis Tool for Hierarchical Real Time Components

Component Based Software Design

Professors
Tommaso Cucinotta
Luca Abeni
Marco Di Natale
Mauro Marinoni

Student
Silvio Bacci

Index

1 Introduction

1.1 Introduction

The objective of the project was to extend the schedulability analysis tool for hierarchical real-time EDF and FP components developed by the Professor Luca Abeni. The extension aims to add to the tool an analysis in case of EDF and multi-CPU partitioned scheduling. In addition the tool was also equipped with an useful research tool used to find a periodic server able to schedule a specific taskset.

1.2 Project requirements

The original project requirements are reported hereafter:

“Extend Abeni's schedulability analysis tool for hierarchical RT EDF/FP components to EDF and Multi-CPU partitioned case”

In the following chapter we are going to explain all the theoretical results regarding:

- Schedulability analysis for periodic tasks
- Schedulability analysis for Hierarchical components
- Schedulability analysis for multi-CPU partitioned case

2 Schedulability Analysis for Periodic Tasks

2.1 Introduction

In this chapter we are going to analyse the main theoretical results useful to conduct a schedulability analysis on a given taskset. In particular we are going to explain how we used these results in order to realize the tool. Let us consider separately the case of fixed and dynamic priority analysing the different results in case of constrained deadline.

2.2 Real-time concepts

First of all we need to introduce some some formal definition used below. Let us define a taskset T as a set of tasks τ_i, where each task can be expressed according to the **(C, D, T) model** as:

$$\tau_i = (C_i, D_i, T_i)$$

where C_i is the computational time, D_i is the deadline and T_i is the period. For what concerns the scheduling algorithms we are going to consider just the following cases:

- **FP**
  - Fixed priority algorithm in the general sense (task order chosen off-line by the user)
- **RM**
  - Fixed priority algorithm in which the tasks are ordered by increasing periods
- **DM**
  - Fixed priority algorithm in which the tasks are ordered by increasing deadlines
- **EDF**
  - Dynamic priority algorithm

2.3 Taskset with deadlines equal to periods

In many cases it is useful to have a very simple test to see if the taskset is schedulable. In particular we are going to explain the main approaches and tools used to check schedulability.

2.3.1 Utilization bound analysis

A sufficient (and necessary in case of EDF) test is based on the utilization bound. The utilization least upper bound (U_{lub}) for a scheduling algorithm A is the smallest possible utilization U_{lub} such that, for any taskset T , if the taskset's utilization U is not greater than U_{lub} (i.e. $U \leq U_{lub}$), then the task set is schedulable by algorithm A . Therefore the schedulability test consist in computing the taskset's utilization as:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Then if $U \leq U_{lub}$ the taskset is schedulable, if $U > 1$ the taskset is not schedulable while if $U_{lub} < U \leq 1$, the taskset may or may not be schedulable. In general no scheduling algorithm can schedule a task set if $U > 1$, so U_{lub} must be less than or equal to one and in particular an algorithm A is optimal if its $U_{lub} = 1$. Let's try to find out the different values for U_{lub} according to different scheduling algorithms.

Theorem (Liu and Layland, 1973):

Given a task set of periodic (or sporadic) tasks, with relative deadlines equal to periods, the task set is schedulable under RM if

$$U \leq U_{lub} = n \cdot \left(2^{1/n} - 1\right)$$

□

If we consider the a taskset of periodic (or sporadic) tasks, with relative deadlines less than periods, we can define another taskset's utilization that is:

$$U' = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

Then the test is the same as the one for RM (or DM), except that we must use *U'* instead of *U.*

Theorem:

Given a task set of periodic or sporadic tasks, with relative deadlines equal to periods, the task set is schedulable by EDF if and only if

$$U \leq U_{lub} = 1$$

□

Note that in this case we have a necessary and sufficient condition. This means that for what concerns the fixed priority algorithms we need something more to achieve a necessary condition.

2.3.2 Workload analysis

As we have already seen, we have not a sufficient and necessary condition in case of fixed priority scheduling algorithms in general. The basic idea is to introduce a sufficient and necessary condition based on the workload of the taskset. First of all let us introduce the workload function that is:

$$W_i(t) = C_i + \sum_{j<i} \lceil \frac{t}{T_j} \rceil C_j$$

Theorem (Lehoczky, Sha and Ding, 1989):

A set of fully preemptive periodic tasks can be scheduled by a fixed priority algorithm if and only if

$$\forall i = 1, \dots, n \qquad \exists t \in (0, D_i] \qquad : \qquad W_i(t) \leq t$$

□

Later, Bini and Buttazzo restricted the number of points in which the condition has to be checked to the following testing set:

$$TS_i = P_{i-1}(D_i)$$

where $P_i(t)$ is defined by the following recurrent expression:

$$\begin{cases} P_0(t) = t \\ P_i(t) = P_{i-1}(\lfloor \frac{t}{T_i} \rfloor T_i) \cup P_{i-1}(t) \end{cases}$$

Theorem (Bini and Buttazzo, 2004):

A set of fully preemptive periodic tasks can be scheduled by a fixed priority algorithm if and only if

$$\forall i=1,\dots,n \qquad \exists t \in TS_i \qquad : \qquad W_i(t) \le t$$

□

2.3 Taskset with deadlines less than periods

In this case the workload analysis is still valid and so we have already a necessary and sufficient condition for fixed priority algorithms. The problem in this case regards EDF because we have no more an useful utilization bound test.

2.3.1 Processor demand approach

For what concerns EDF we need to introduce a new necessary and sufficient test called processor demand approach. First of all let us introduce the demand bound function that is:

$$dbf(t)=\sum_{i=1}^{n} \lfloor \frac{t+T_i-D_i}{T_i} \rfloor C_i$$

Theorem:

A set of synchronous periodic tasks with relative deadlines less than or equal to the periods can be scheduled by EDF if and only if U < 1 and

$$\forall t \in D \qquad dbf(t) \le t$$

where

$$D=\{d_k \mid d_k \le min[H, max(D_{max}, L^*)]\}$$

and

$$L^*=\frac{\sum_{i=1}^{n}(T_i-D_i)U_i}{1-U}$$

□

As we can see this approach can be applied both in case constrained and non-constrained deadlines. So we will use the processor demand approach to test the schedulability of a taskset in both the cases (as we do with the workload analysis for fixed priority algorithms).

2.4 Summary

Just for recap, let us consider the following table:

	D = T	**D < T**
Fixed Priority	if: utilization bound iff: workload analysis	iff: workload analysis
EDF	iff: utilization bound / processor demand approach	iff: processor demand approach

2.5 Implementation

In the final implementation the schedulability of a taskset under fixed priority is checked using the workload analysis while under EDF using the processor demand approach. In particular we are going to explain the libraries and source files created. The first important library introduced is called "*dbf.c/h*" in which we can find the functions able to compute the workload and processor demand functions (the name of the file will be better explained in the next chapter). The second and more important library is called "*s_analysis.c/h*". Inside this file we can find two important functions: *s_analysis_fp* and *s_analysis_edf*. These functions are the simply a C implementation of the workload analysis (*s_analysis_fp*) and the processor demand approach (*s_analysis_edf*). Finally we can find also a source file called "*s_analysis.c*" from which we call the previous library to check the schedulability of a taskset specified in a given file.

3 Schedulability Analysis for Hierarchical Components

3.1 Introduction

In this chapter we are going to analyse the main theoretical results useful to conduct a schedulability analysis on a given component. First of all we need to introduce some concepts about scheduling of components. In the general case an application is composed by a set of components within which we can find a set of tasks (i.e. taskset). The main idea is to use a two-level hierarchical scheduling system, in which we have two schedulers:

- Host (or global, root) scheduler for scheduling components
- Local (or 2nd level) scheduler for scheduling tasks inside each component

This can be easily done considering components isolated thanks to different VMs. This means that the global scheduler will schedule just VMs (or components) while the local scheduler will be the scheduler running inside the VM.

3.2 Demanded and supplied time

Simplifying a little bit the problem, we can test the schedulability of a component knowing the time needed by the component to respect its temporal constraints and the amount of time provided by the global scheduler. In fact a component will be schedulable if:

$$\text{demanded time} \quad \leq \quad \text{supplied time}$$

where the demanded time is the amount of time (in a time interval) needed by a component while the supplied time is the amount of time (in a time interval) given by the global scheduler to a component.

3.2.1 Demanded time

The good new in this case is that the demanded time needed by the component can be easily computed using the processor demand function (for EDF) and the workload function (for FP).

$$dbf(t)=\sum_{i=1}^{n}\lfloor\frac{t+T_i-D_i}{T_i}\rfloor C_i \quad \textit{(EDF)}$$

$$W_i(t)=C_i+\sum_{j<i}\lceil\frac{t}{T_j}\rceil C_j \quad \textit{(RM)}$$

3.2.2 Supplied time

For what concerns the supplied time we will consider the case of a periodic server. This means that the idea is to use a periodic server to schedule the components (e.g. the global scheduler could be a constant bandwidth server). Remembering that a periodic server is described by the budget Q_s and by the period T_s, the supplied bound function can be expressed as:

$$sbf(t)=\begin{cases} 0 & if \quad t < 2\cdot(T_s-Q_s) \\ (n-1)\cdot Q_s & if \quad n\cdot T_s-Q_s \leq t < (n+1)\cdot T_s-2\cdot Q_s \\ t+n\cdot Q_s-(n-1)\cdot T_s & if \quad (n+1)\cdot T_s-2\cdot Q_s \leq t < (n+1)\cdot T_s-Q_s \end{cases}$$

3.3 Schedulability analysis

Theorem (fixed priority):

A component (i.e. a set of periodic tasks with relative deadlines less than or equal to periods) characterized by a fixed priority local scheduler is schedulable by the global scheduler if and only if

$$\exists t\in TS_i\cup SBF_i \quad : \quad dbf(t)\leq sbf(t)$$

where TS_i *is the testing set introduced during the workload analysis explanation (chapter 2) and* SBF_i *is the testing set given by all the points in which the supplied bound function starts to become flat, i.e.:*

$$SBF_i=\{t\,|\,t=n\cdot Ts-Qs,\, t<max\{TS_i\}\}$$

where Q_s *and* T_s *are respectively the budget and the period of the periodic server.*

□

Theorem (EDF):

A component (i.e. a set of periodic tasks with relative deadlines less than or equal to periods) characterized by a EDF local scheduler is schedulable by the global scheduler if and only if:

$$\forall t \in D \cup SBF \quad : \quad dbf(t) \leq sbf(t)$$

where D *is the testing set introduced during the processor demand approach explanation (chapter 2) and* SBF *is the testing set given by all the points in which the supplied bound function starts to become flat, i.e.:*

$$SBF = \{ t \mid t = n \cdot Ts - Qs \, , t < max\{D\} \}$$

where Q_s *and* T_s *are respectively the budget and the period of the periodic server.*

□

3.4 Implementation

In the final implementation the schedulability of a taskset is checked using the result of the last two theorems. In particular we are going to explain the libraries and source files created. The first important library introduced is called "*sbf.c/h*" in which we can find the function able to compute the supplied bound function for a given periodic server. The second library is called "*h_analysis.c/h*". Inside this file we can find two important functions: *h_analysis_fp* and *h_analysis_edf*. These functions are the simply a C implementation of the last two theorems. In particular the idea is to use the functions contained in the "*dbf.c/h*" library (described in the previous chapter) to compute the workload and processor demand functions and compare them to the specified periodic server. Finally we can find also a source file called "*h_analysis.c*" from which we call the previous library to check the schedulability of a taskset specified in a given file.

4 Search Periodic Server

4.1 Introduction

In this chapter we are going to explain how we can find a specific periodic server able to schedule a given taskset. Of course the objective in this case is to find the parameters of the server, i.e. budget Qs and period Ts.

4.2 Algorithm

The idea behind the periodic server research is to try a subset if all the possible periodic servers and take the best one. Of course we need to specify what means best in this case. In general the tool aims to find periodic server, able to schedule a taskset, with the minimum bandwidth. This means that we want to find the smallest periodic server that can solve our problem.
In order to performing this research we test the server changing the parameters. In particular the ranges used are:

- $Ts \in [min\{T_i\}, 2 \cdot max\{T_i\}]$
- $Qs \in (0, Ts \cdot BW]$ where BW is the current best bandwidth

Every time we change the parameters we take note of the best periodic server, i.e. the one with the minimum bandwidth.

4.3 Implementation

In the final implementation we can find a function called *find_periodic_server* in the library "*h_analysis.c/h*". This function performs a two nested for cycles in which we test all the period and budget as explained above. Finally we can find also a source file called "*find_ps.c*" from which we call the previous library to find a periodic server for a taskset specified in a given file.

5 Schedulability Analysis for M-CPU partitioned scheduling

5.1 Introduction

In this chapter we are going to analyse the main theoretical results useful to conduct a schedulability analysis on multi-CPU machine. First of all we need to introduce some concepts about the possible scheduling approaches in case of multi-processor machines. In general we can can have two approaches:

- Partitioned scheduling
- Global scheduling

In the global scheduling case (the one we are going to ignore in this project), the idea is to use a single task queue shared by M CPUs and select the first M ready tasks. On the other side in the partitioned scheduling the idea is to reduce the problem assigning statically tasks to M CPUs and then performing uniprocessor scheduling for each CPU. The good new using the partitioned scheduling is that we can use all the tools presented in the previous chapters to check the schedulability of the tasks.

5.2 Bin-packing

The real problem with partitioned scheduling is to find an optimal way to assign tasks to the different cpus. In fact this problem, also known as bin-packing, is a NP-hard problem in the general sense and so we have not an optimal solution. However, in the literature we can find lots of different heuristic bin-packing algorithms useful for our purposes:

- **Best fit**
  - Assign each task to the processor with the smallest empty space.
- **First fit**
  - Assign each task to the first processor that can schedule it.
- **Next fit**
  - Assign each task to the same processor used for the previous task. If the new task can be scheduled together with the tasks already present continue, otherwise assign the task to a next empty cpu.
- **Worst fit**
  - Assign each task to the processor with the largest empty space, otherwise assign the task to a next empty cpu.
- **First fit decreasing**
  - Sort tasks in decreasing order of utilization, then assign tasks according to the first fit

Let us define M_{on} as the number of bins used by an online algorithm and M_o the optimal number of bins. Any online algorithm uses at least 4/3 times the optimal number of bins:

$$M_{on} \geq \frac{4}{3} M_o$$

Is addition some useful results are:

- NF and WF never use more than $2M_o$ bins.
- FF and BF never use more than $\frac{17}{10} M_o + 1$ bins.
- FFD never uses more than $\frac{11}{9} M_o + 4$ bins.

5.3 Utilization under fixed priority

The heuristic algorithms explained before are useful and widely used for scheduling task under EDF. The problem borns when we deal with fixed priority algorithms. In fact in this case we can use of course once again solutions like "*First fit*" and "*Next fit*" but we cannot use, or at least in this version, the other algorithms. The problem is related to the utilization factor computed as:

$$U = \sum_i \frac{C_i}{T_i}$$

If we think about the previous relation we can understand that it has no sense for fixed priority. So the solution is to use another kind of utilization. Remember that the schedulability check under fixed priority says that we need to verify:

$$\forall i = 1, \ldots, n \qquad \exists t \in (0, D_i] \qquad : \qquad W_i(t) \leq t$$

If we use this result we can compute a new utilization factor as:

$$U_i = \frac{W_i(t)}{t}$$

Moreover the general utilization of the entire taskset will be:

$$U_i = max_\tau \{ min \{ \frac{W_i(t)}{t} \} \}$$

This means that for every task we are searching the minimum possible value (note that some values could be also greater than 1). Then among all the tasks, we are going to take the maximum value that will be always less than or equal to one (if the taskset is schedulable).

At the end we can use this utilization for what concerns the algorithms "*Best fit*" and "*Worst fit*". The only algorithm we cannot use under fixed priority is "*First fit decreasing*" because we cannot order the tasks in a different way from the one chosen fixed.

5.4 Implementation

In the final implementation the schedulability of a taskset in a multi-CPU environment is checked using the schedulability checking tools explained in the previous chapters and the bin packing algorithms. In particular we are going to explain the libraries and source files created. The first important library introduced is called "*mcpu_analysis.c/h*" in which we can find the function able to perform the desired bin packing algorithm. As already said it uses different ways to compute the utilization of the taskset according to the scheduling algorithm (fixed priority of EDF). The schedulability on a specific CPU is checked using the "*h_analysis*" tool if a periodic server is specified for the CPU. Moreover the function is also able to find a periodic server if no one is specified. In this case the schedulability check is perform using the "*s_analysis*" tool. Finally we can find also a source file called "*mcpu_analysis.c*" from which we call the previous libraries to check the schedulability of a taskset on a specific vm, both of them specified in a given file.

6 Implementation

6.1 Introduction

In this chapter we are going to explain how the tool was organized and developed. First of all is important to note that the tool was developed using the C programming language and compiled thanks to the gcc compiler installed on machine running an OS Linux distribution Debian 8 (Jessie). The idea behind the organization of the files is to divide the code in order to obtain reusable components able to works also in completely different systems.

6.2 Project files

The project folder is composed by the following folder:

- bin
- lib
- src
- taskset
  - Contains the files in which are defined the different taskset used during testing
- vm
  - Contains the files in which are defined the different vm used during testing
- report
  - Contains the report of the project

The *bin* folder is used to store the executable files obtained after the compilation and linkage of the source files. The src folder contains the source files able to perform the various schedulability analysis presented. In particular we can find the following programs:

- sbf
  - Input: budget and period of the periodic server
  - Output: compute the supplied bound function in a given interval of time ($t \in [0, 100]$)
- s_analysis
  - Input: taskset and scheduling algorithm
  - Output: check schedulability of the taskset under the specified scheduling algorithm
- h_analysis
  - Input: taskset, budget and period of the periodic server and scheduling algorithm
  - Output: check schedulability of the taskset under the specified scheduling algorithm with the specified periodic server
- find_ps_server
  - Input: taskset and scheduling algorithm
  - Output: search the best periodic server able to schedule the taskset under the specified scheduling algorithm

- mcpu_analysis
  - Input: taskset, vm, scheduling algorithm and allocation algorithm
  - Output: check schedulability of the taskset considering the parameters of the VMs (for all of them run the same specified scheduling algorithm) and the specified allocation algorithm

The *lib* folder is used to store the library (i.e. reusable components) and we can find the following organization:

- task
  - task_io.c/h
  - taskset.c/h
  - sorting.c/h
  - utilities.c/h
  - structs
    - task.h
    - taskset.h
    - periodic_server.h

- vm
  - vm_io.c/h
  - sorting.c/h
  - utilities.c/h
  - structs
    - cpu.h
    - vm.h

- schedulability
  - dbf.c/h
  - sbf.c/h
  - s_analysis.c/h
  - h_analysis.c/h
  - mcpu_analysis.c/h

In the *task* folder we can find all the useful libraries to load and manage the taskset. For example we can find all the functions useful to compute the taskset's utilization, max period, sort taskset according to different policies and so on.
In the *vm* folder we can find the libraries useful to load and manage the vm. In particular we can find all the functions useful to sort the different CPU according to different policies.
In the *schedulability* folder we can find the main libraries of the project. In particular in each file we can find the schedulability analysis as described in the previous chapter.

7 Testing

7.1 Introduction

In this chapter we are going to report all the testing results.

References

[1] L. Abeni, *Hierarchical Scheduling for Components*. Slides for Component Based Software Design course at Scuola Superiore Sant' Anna, May, 2018.

[2] L. Abeni, *Supplied Time for a Periodic Server*. Slides for Component Based Software Design course at Scuola Superiore Sant' Anna, May, 2018.

[3] L. Abeni, *Again on Supplied Bound Function*. Slides for Component Based Software Design course at Scuola Superiore Sant' Anna, May, 2018.

[4] G. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*. 3rd ed., Springer, 2011.

[5] C. Lu, *Real-Time Scheduling: Single Processor*. Slides at Washington University of St Louis.

[6] G. Lipari, Fixed priority *scheduling*. Slides for Real-Time Systems course at Scuola Superiore Sant' Anna, November, 2011.

[7] G. Lipari, *EDF scheduling*. Slides for Real-Time Systems course at Scuola Superiore Sant' Anna, November, 2011.